TEXAS INSTRUMENTS

The proLogic™ Compiler

Release Notes

Release 2.10

The proLogic™ Compiler

Release 2.10

IMPORTANT NOTICE

Texas Instruments (TI) provides this software **AS IS** without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. TI may discontinue, make improvements to and or change the program(s) described herein at any time and without notice. TI does not warrant that this software package will function properly in every software/hardware environment.

Users are authorized to copy, duplicate, and distribute this product for internal use only without further permission from TI provided that the copyright is maintained on each copy. Selling of this product without prior consent in writing from TI is prohibited.

proLogic compiler, Release 2.10

This logic compiler was prepared by proLogic Systems Inc. for distribution by Texas Instruments Incorporated, expressly to support devices manufactured by them.

Assistance on programming Texas Instruments Programmable Logic is also available, upon request, from the nearest TI field sales office, local authorized TI distributor, or by calling the Texas Instruments PLD Hotline (214) 997-5666, or Texas Instruments PLD Bulletin Board Service (214) 997-5665. [BBS Information: 14.4K, 8, N, 1]

TRADEMARKS

proLogic is a trademark of proLogic Systems Inc..
IBM is a trademark of International Business Machines Corporation.
PALASM is a trademark of Advanced Micro Devices, Inc.
ALES and **ALS** are trademarks of Actel Corporation.
View*logic*, Viewdraw, and Viewfile are trademarks of View*logic* Systems, Inc.

Section 1

Introduction

Introduction

This release note documents features and limitations of the proLogic™ compiler release 2.10. The proLogic compiler is a logic synthesis tool for Texas Instruments (TI) Programmable Logic Devices (PLD). Release 2.10 enables PLD designers to migrate PLD design files to Field Programmable Gate Arrays (FPGA). Also included in release 2.10 are additional features to release 2.00. Appendix A is a tutorial on how to use the proLogic compiler to design with Texas Instruments Field Programmable Gate Arrays.

The proLogic Compiler Release 2.10 Package Contains:

- proLogic Compiler Release Notes, Release 2.10
- One proLogic Diskette, Release 2.10
- proLogic Compiler User's Guide, Release 2.00

Document Contents

Section 2

Installation

Installation

Minimum system requirements is an IBM™ PC or clone with a hard drive with 1.8M bytes of space available for file storage.

To install proLogic release 2.10, the following steps are necessary:

Step 1: Create a proLogic directory on your hard drive using the DOS MD command.

```
MD PROLOGIC
```

Step 2: Change directories to your proLogic directory.

```
CD PROLOGIC
```

Step 3: Copy PROLOGIC.EXE to the hard drive.

```
COPY a:\PROLOGIC.EXE
```

Step 4: Execute PROLOGIC.EXE file to extract proLogic files.

```
PROLOGIC  -D
```

Step 5: Operate the proLogic compiler as you normally do referring to the Release 2.00 User's Guide and 2.10 Release Notes. Should problems arise, contact the TI FPL Helpline at (214) 997-5666.

Section 3

New Features

Migrating proLogic Design Files to FPGAs

For many of today's digital system design questions, the solution is Texas Instruments Field Programmable Gate Arrays. In the past these have been solved with Texas Instruments PLDs and EPLDs both capable of being designed proLogic. However the problem arose, how do you transfer yesterday's proLogic PLD solution done with programmable logic solutions to today's FPGA solution. The answer is now here, proLogic release 2.10. This new version allows this conversion by a simple command line option entered when you perform LC command.

The new command line option to create a `.pds` file for Logic Enhancer/Synthesizer (ALES™) processing is `-p`.

The proper syntax is of the following:

```
lc <design_name> -p
```

Of course this can be used with any of the other options that the `'lc'` command allows. The purpose of this option is to allow the user to create a ALES input file using proLogic design software thus completing loop of design software that can be used in conjunction with Texas Instruments Field Programmable Logic Devices.

The next step in the FPGA flow would to activate ALES which would be done by the following command:

```
alesl -pds <design_name>
```

After running ALES on the design file, you need to create a symbol to be used with Workview, and add the ALSPDS attribute to that symbol. You must also create the WIR file which is used in further processing of the design (simulation and makeadl). The proper syntax for this command is:

```
edifneti <design_name> -o <path of \designs\wir dir>
```

When this completes, the WIR file will be correctly placed and further FPGA design processing can occur.

New Include File Features

Two new functions have been added to the proLogic logic compiler. Both enhance proLogic designs using the TIBPLS506A and TIBPSG507A devices.

File: ATINT.H

Purpose: To aid in the definition of internal registers of the TI sequencer family.

Usage:

```
include atint;   /* utilize new internal register definition
                                                       aid */

INT   count ^ (4..0); /* define internal registers for
                                                   counter */
```

For a full scale application see example under RSCNTR.H.

Related Files: AT.H

File RSCNTR.H

Purpose: To aid in the design of counters using the RS flip-flops of the TI sequencer family.

Usage:

```
title { this is included as rs.pld }

include a506;
include rscntr;
include at;
include atint;   /* utilize new internal register definition
                                                       aid */
define count = (cnt\4..0);
INT   count ^ (4..0);    /* define internal registers for
                                                   counter */
PIN clk @ 1;
PIN up  @ 2;

if (up)
  ++ count ;
else
  -- count ;
```

Related Files: DCNTR.H, TCNTR.H

New Devices Supported

Due to the popularity of the TIBPAL22V10 and the PLCC package, we have released the PLCC version of the TIBPAL22V10. It can be utilized by including the `'p22v10c.h'` file were the `'p22v10.h'` was normally used. Pin to pin conversions can be calculated by comparing the DIP package to the PLCC package and using the new names or by looking at the `'p22v10c'` diagram.

Section 4

Limitations

Current Limitations

The proLogic compiler has the following limitations when using the `-p` option:

1. Only the `p16XX, p20XX, p22v10, p22v10c`, and `p22vp10` devices can be utilized at this time. EPLD series, sequencer devices and a generic model will be supported on a later version.

2. JEDEC files can not be generated at the same time as the .PDS file. If the JEDEC file is desired for simulation purposes rerun without the `-p` option and the JEDEC file will be created.

3. The `'@'` operator can not be used for pin assignments. Currently the desired signal name is converted to `'pinxx'` for a signal name. This is more commonly used in EPLD designs but possibly could be used in standard PLD design files.

4. The synchronous preset on the 22v10 family of devices (`p22v10, p22vp10,` and `p22v10c`) is currently assigned to the asynchronous preset term in the `.PDS` file. Remember this when utilizing the synchronous preset line in your 22V10 applications.

LS Simulation Errata

Simulating With Different Clocks on the Same Device (EP1830 & EP630):

If you are using different clock pins on the same device you may see some different results than expected during logic simulation. The LS logic simulator included with proLogic package reads the 1,0,X inputs for the device then will clock all pins indicated one at a time starting with pin 1. If your application depends on two clock pins switching at the same time, you should consider the asynchronous clock feature of these devices to have JEDEC that simulates and can be used for functional test on a programmer.

Using the Power Up/Down Feature

While this may be used for power up/down testing during proLogic simulation. This feature should not be expected to yield desired effects during functional vector testing after programming because it is not supported in JEDEC. If you desire a JEDEC

file to use during simulation and programming with functional test, do not implement this feature in the test vector section of your `.PLD` file.

How to use Preload Vectors

The logic level value that is placed on the output pin during preload, needs to be the opposite of what the value the register is to contain when the preload is finished. For example if a logic 1 is to be placed in the register, a low level needs to be placed on the output pin during the preload. The update to the `LS.1` file corrects the preload value problem that was unnoticed until a test vector set with preload vectors was used on a programmer. Any test vectors with preloads generated with proLogic will now work correctly on any JEDEC programmer.

Test Vector Use of Asynchronous Clear on EP630 and EP1830

Currently the simulation of asynchronous clears on the EP630 and the EP1830 is not available. The logic is properly compiled, however the simulator does not work with this feature.

Appendix A

How To Use prologicTo Design With Texas Instruments Field Programmable Gate Arrays

Introduction

There are two approaches to design an electronic circuit: schematic entry and logic synthesis.

The proLogic™ compiler is a logic synthesis tool for Texas Instruments (TI) Programmable Logic Devices (PLD). This tool is now extended to enable PLD designers to use the familiar Boolean equations and state machine entry for designing with Texas Instruments Field Programmable Gate Arrays (FPGA).

The proLogic compiler generates in addition to the JEDEC file a PALASM™ 2 compatible file in PDS format which is used as input for the Logic Enhancer/Synthesizer (ALES™) and the Action Logic System (ALS™) to design and program a Texas Instruments FPGA. All the necessary softwares and hardware can be provided by Texas Instruments as a complete FPGA logic synthesis design kit.

Using a simple example, this tutorial shows the complete design flow how to use proLogic, ALES and ALS to design and program a circuit with FPGA. The tutorial shows all the commands as they are required to implement the design. However, you might need to consult the manuals of each software package if you need more detailed information. The ALES and ALS software used are the latest version 2.11. If you still have an old version, please contact Texas Instruments for update.

Design Flow

The FPGA design flow using proLogic consists of the following steps:

- Divide the design into function blocks.
  Use proLogic syntax to describe the function of each block.
  Use proLogic simulator to verify the functionality of each block.
  Use proLogic to synthesize the logic for standard PLD gates/flip-flops and generate a PDS file.
- Use ALES to map the logic to the FPGA complex gates/flip-flops and generate an EDIF netlist (technology mapping).

- ❑ Use a CAD tool, e.g. Viewlogic™, to read the EDIF netlist and create a symbol for each block.
  Use the same CAD tool to create a top level schematic which can consist of one or several symbols plus the input/output buffers and generate a netlist for the complete design.
  The Viewlogic schematic entry tool is included in every ALS package.
- ❑ Convert the CAD netlist of the complete design to an FPGA netlist (ADL format).
  Use ALS to select an FPGA device and to place & route.
  Use the ALS Timer for timing analysis.
  Generate an FPGA fuse map and use an Activator to program the FPGA.

A Design Example

The example used in this tutorial (CNT4L) is a simple 4-bit counter with synchronous clear. The counter is reset to zero when CLR = 0 and it counts upwards when CLR = 1.

proLogic Flow

Create the proLogic Source File, Verify Functionality and Generate a PDS File

Step 1: Create a Subdirectory for the proLogic Application Files

```
md \designs\CNT4L
```

Step 2: Create the proLogic Source File With an ASCII Editor

```
File name:   \designs\CNT4L\CNT4L.pld

Enter the following text:
```

```
title {         Filename:       CNT4L.pld
                Function:       simple 4-bit binary counter
                Designer:       Dung Tu
                Date:           1.Sep.92
                Note:           22V10 is specified as device to produce a
                                PDS file and for simulation. The design is
                                later converted to FPGA using ALES.
}

include p22v10;     /* Specify a PLD device type                  */
include XOR.H;      /* To implement the "%" operator as Exclusive OR */

/* Input pins */

define  CLK = pin1;
define  CLR = pin2;

/* Output pins */

define  Q0 = pin14;
define  Q1 = pin15;
define  Q2 = pin16;
define  Q3 = pin17;

/* Specify register outputs of the 22V10 as active high  */

Q0 = q;
Q1 = q;
```

```
Q2 = q;
Q3 = q;

/* The generic format for the n-th bit (Qn) of an n-bit counter is:  */
/* Q  = Q      %   (Q    & Q    & ... & Q )
/*  n    n-1         n-1    n-2          0                            */
/* proLogic uses "Q.d" for a flipflop input and "Q.q" for the output */
/* Equations for a 4-bit counter:                                     */

Q0.d = !Q0.q & CLR;
Q1.d = (Q1.q %  Q0.q) & CLR;
Q2.d = (Q2.q % (Q1.q & Q0.q)) & CLR;
Q3.d = (Q3.q % (Q2.q & Q1.q & Q0.q)) & CLR;

/* Test vectors for proLogic functional simulator */

test_vectors
{    CLK   CLR    Q3    Q2    Q1    Q0 ;    /* count */

     C     0      L     L     L     L  ;    /* clear */

     C     1      L     L     L     H  ;    /* 1 */
     C     1      L     L     H     L  ;    /* 2 */
     C     1      L     L     H     H  ;    /* 3 */
     C     1      L     H     L     L  ;    /* 4 */
     C     1      L     H     L     H  ;    /* 5 */
     C     1      L     H     H     L  ;    /* 6 */
     C     1      L     H     H     H  ;    /* 7 */
     C     1      H     L     L     L  ;    /* 8 */
     C     1      H     L     L     H  ;    /* 9 */
     C     1      H     L     H     L  ;    /* 10 */
     C     1      H     L     H     H  ;    /* 11 */
     C     1      H     H     L     L  ;    /* 12 */
     C     1      H     H     L     H  ;    /* 13 */
     C     1      H     H     H     L  ;    /* 14 */
     C     1      H     H     H     H  ;    /* 15 */
     C     1      L     L     L     L  ;    /* 0 */
}
```

The source file begins with a "`title`" block where you can add the design name and comments. The title block must be enclosed between a "{" and a "}".

The "`include`" part is used to specify a PLD device type. This is required so that a JEDEC file can be generated which is used later by the proLogic simulator to verify the functionality of the design using some test vectors.

The "`include`" part consists also of the header files which are required for a correct function of the compiler. The header file `XOR.H` is required, for example, to implement the operator "%" as Exclusive OR.

The pin definition part refers to the proLogic diagrams for each PLD device type (see proLogic manual) and replaces the pin number by a more meaningful signal name using the "`define`" statement. This pin definition is required only for the simulation and due to the compatibility to the PALASM2 syntax. The real pin assignment for FPGA is done later with the ALS development software.

Because the register outputs of the 22V10 can be configured as active-high or active-low they must be specified so that proLogic compiles the design correctly. `Q0 = q` specifies active-high, `!Q0 = q` specifies active-low outputs.

The equation part defines the logic of the design. This part can be also a state machine or a truth table.

The test vectors part is used by the proLogic simulator to verify that the logic functions defined for the specified PLD are correct. The test vectors define the inputs to

the PLD and specify the output expected. The test patterns must be enclosed between a "{" and a "}".

The convention is as follows:

```
0 - drive input LOW
1 - drive input HIGH
L - test output LOW
H - test output HIGH
C - drive input LOW-HIGH-LOW
```

Step 3: Create The `.PDS` File and the JEDEC File

To create the `.PDS` file, use the "`lc`" command with the option `-p`:

```
lc CNT4L -i\prologic -p
```

The option `-i\prologic` tells proLogic to find the include and header files in the `\prologic` executable file directory.

To create the JEDEC file, enter the command `lc` without `-p`:

```
lc CNT4L -i\prologic
```

If you create the following batch file `(pl.bat)`:

```
lc %1 -i\prologic -p
lc %1 -i\prologic
```

you need only to enter the command:

```
pl CNT4L
```

The `CNT4L.PDS` file which proLogic generates is as follows:

```
CLK CLR nc nc nc nc nc nc nc nc nc GND nc Q0 Q1 Q2 Q3 nc nc nc nc nc nc VCC

EQUATIONS

Q0 :=
          /Q0 * CLR ;

Q1 :=
           Q1 * /Q0 * CLR
         + /Q1 * Q0 * CLR ;

Q2 :=
           /Q2 * Q1 * Q0 * CLR
         + Q2 * /Q0 * CLR
         + Q2 * /Q1 * CLR ;

Q3 :=
           /Q3 * Q2 * Q1 * Q0 * CLR
         + Q3 * /Q0 * CLR
         + Q3 * /Q1 * CLR
         + Q3 * /Q2 * CLR ;

Q3.TRST =
           VCC ;

Q2.TRST =
           VCC ;

Q1.TRST =
           VCC ;
```

```
Q0.TRST =
          VCC ;
```

Step 4: Functional Simulation

To use the proLogic simulator for design verification, enter the following command:

```
ls CNT4L -a\prologic\p22v10.lxa
```

The option `-a` tells proLogic to use the architecture description file `p22v10.lxa` in the directory `\prologic`. The simulator uses this file and the JEDEC file to compile an optimized device model prior to the actual simulation.

The result is written into the following `CNT4L.tst` file:

```
proLogic Simulator
Texas Instruments V2.0
Copyright (C) 1991 Prologic Systems

Architecture Description: \prologic\p22v10.lxa
JEDEC Fuse Information:   CNT4L.jed
JEDEC Test Vectors:       CNT4L.jed

V01    C0NN NNNN NNNN NLLL LNNN NNNN
V02    C1NN NNNN NNNN NHLL LNNN NNNN
V03    C1NN NNNN NNNN NLHL LNNN NNNN
V04    C1NN NNNN NNNN NHHL LNNN NNNN
V05    C1NN NNNN NNNN NLLH LNNN NNNN
V06    C1NN NNNN NNNN NHLH LNNN NNNN
V07    C1NN NNNN NNNN NLHH LNNN NNNN
V08    C1NN NNNN NNNN NHHH LNNN NNNN
V09    C1NN NNNN NNNN NLLL HNNN NNNN
V10    C1NN NNNN NNNN NHLL HNNN NNNN
V11    C1NN NNNN NNNN NLHL HNNN NNNN
V12    C1NN NNNN NNNN NHHL HNNN NNNN
V13    C1NN NNNN NNNN NLLH HNNN NNNN
V14    C1NN NNNN NNNN NHLH HNNN NNNN
V15    C1NN NNNN NNNN NLHH HNNN NNNN
V16    C1NN NNNN NNNN NHHH HNNN NNNN
V17    C1NN NNNN NNNN NLLL LNNN NNNN

No errors detected with 17 Test Vectors.
```

The test result file shows that no errors was detected so that the logic equations work correctly.

ALES Flow

Convert The PDS File To An FPGA Edif Netlist

To map the PDS logic equations to the FPGA hardmacros and generate an EDIF 2.0.0 netlist (`CNT4L.edn`), enter the following command:

```
ales1 -pds CNT4L
```

ALES selects per default the TPC10 series (TPC1010, TPC1020). To map the design to the TPC12 series (TPC1225, TPC1240, TPC1280), use the following command:

```
ales1 -pds fam:act2 CNT4L
```

Viewlogic Flow

Create Block Symbols — A top level schematic and generate a netlist including input/output buffers for the complete design.

The equations you created describe the function of the 4-bit counter. However, these equations do not include the FPGA input and output buffers.

To complete the design, it is required that you use a CAD tool to create a symbol having the same name as your equation file (CNT4L as in this example) and then create a top level schematic (CNT4LT) which includes the symbol and the required input/output buffers.

The following steps are a proposal how to create a directory structure for Viewlogic which works nicely with proLogic and ALES.

All Viewlogic projects are assumed to be saved under a directory \proj. For every individual design you use the Viewfile™ utility to create the required subdirectories so that your CNT4LT design will be stored under `\proj\CNT4LT`.

The directory for proLogic and ALES projects is \designs. Your CNT4L proLogic files are saved under the subdirectory `\designs\CNT4L`. ALS will create later automatically for the top level design CNT4LT the subdirectory `\designs\CNT4LT` where all ALS files are saved.

```
Viewlogic design file subdirectories
 ------------------------------------
\proj\CNT4LT EDIF netlists, simulation netlists,
etc.
\proj\CNT4LT\SCH schematic files
\proj\CNT4LT\SYM symbol files
\proj\CNT4LT\WIR wirelist files

proLogic design file subdirectory
---------------------------------
\designs\CNT4L proLogic files

ALS design file subdirectory
----------------------------
\designs\CNT4LT ALS files, generated by MAKEADL or
ALS
```

Step 1: Create the Project

To invoke Viewlogic enter:

```
wv
```

Use Viewfile to create the project:

```
Window | Open | Viewfile
Project | Create          Project name \proj\CNT4LT
Set | Project             CNT4LT
```

Step 2: Use EDIFNETI to Convert the EDIF Netlist to a Viewlogic Wirelist

Copy the EDIF netlist CNT4L.edn from the subdirectory `\designs\CNT4L` to `\proj\CNT4LT`. To create a wirelist, enter the command:

```
edifneti CNT4L.edn
```

The EDIF netlist reader converts the EDIF netlist to a wirelist and saves this netlist under the subdirectory `\proj\CNT4LT\wir`.

Step 3: Create a Viewdraw™ Symbol

To create a symbol, do the following steps:

```
Window | Open | Viewdraw | Symbol         Symbol name: CNT4L
Change | Block | Sheet | Z-WxH            Block width: 200
                                          Block height: 200
Add | Box                                 Draw a box
Add | Pin                                 Draw the pins
Add | Label                               Name the pins
```

The symbol has two input pins CLR, CLK and four output pins Q0, Q1, Q2, Q3.

```
                ALSPDS   <--- Attribute
               ----------
               |        |
      CLR------|        |-----Q0
               |        |
      CLK------|>       |-----Q1
               |        |
               | cnt4l  |-----Q2       cnt4l is entered by using
               |        |              Add | Text. It is a comment,
               |        |-----Q3       not a label.
               |        |
               ----------
```

To tell ALS that you are working with equations and not with schematic entry, it is required that you add the attribute "ALSPDS" to the symbol you created. To add an attribute, do the following steps:

```
View  | Out              to get more space
Add   | Attr             Enter: ALSPDS
```

The symbol and the wirelist are linked to each other by the names. That is the reason why the symbol name must be identical to the wirelist name.

Step 4: Use Viewdraw to Create a Top Level Schematic and Generate a Top Level Wirelist

To create a top level schematic, do the following steps:

```
Window | Open | Viewdraw | Schematic  Name: CNT4LT
Add | Comp       Name: CNT4L
Add | Comp       Add the input/output buffers
Add | Net        Connect the components
etc...
File|Write       Create top level wirelist
```

Use INBUF as input buffer for CLR, use CLKBUF for CLK, use OUTBUF for the outputs.

You have now generated a complete wirelist which includes your 4-bit counter plus the FPGA input/output buffers and are finished with the Viewlogic flow. The next step is to use ALS to place & route and to generate a fuse map for programing the FPGA.

ALS Flow

Generate ADL Netlist, Specify FPGA Device, Place & Route, Generate Fuse Map

Step 1: Convert Wirelist To ADL Format

To generate the ADL netlist for the CNT4LT design, enter the following commands:

```
cd \proj\CNT4LT
makeadl CNT4LT
```

MAKEADL creates automatically the subdirectory `\designs\CNT4LT` where it saves all the required files for ALS.

Step 2: Invoke ALS and Specify FPGA Device

To invoke the ALS menu, enter the command:

```
als CNT4LT
```

The ALS menu is displayed. To specify an FPGA device, select the following menu items:

```
Project | Device | TPC10 | TPC1010A | 68PLCC
```

This step can only be done within the ALS menu.

Step 3: Run Validate and Place & Route

To run the Validator, select from the ALS menu the following command:

```
Validate | Run
```

The Validator checks whether the FPGA design rules are violated and shows a statistics of the modules used.

To run Place & Route, select:

```
Config | Run
```

Automatic pin assignment is used for this design so that no Pinedit is required.

Step 4: Timing Analysis Using ALS Timer

To do a timing analysis of the design, you can use the ALS "`Timer`". To invoke the Timer, select from the ALS main menu:

```
Timer | Run
```

You can then specify the temperature and voltage by clicking these menu items.

To get the longest delay time, select from the ALS Timer menu:

```
Timing | Longest
```

The "Longest Path Selection" menu is displayed. If you use the default pin sets by clicking OK, the Timer will show you the longest delay paths as following:

```
;  1st  longest path to all endpins
; Rank Total Start pin      First Net     End Net       End pin
;   0   36.4 U1/I8:CLK      IQ1           U1/N11        U1/I0:S
;   1   36.3 U1/I10:CLK     IQ0           U1/N9         U1/I9:S
;   2   29.9 U1/I10:CLK     IQ0           U1/N7         U1/I8:S
;   3   26.1 U1/I10:CLK     IQ0           U1/N6         U1/I8:A
;   4   17.4 U1/I10:CLK     IQ0           IQ0           U1/I10:S
```

The signal names which are displayed on your screen might be different, depending on how you label your signals.

To get all the delay elements which are summed up to the total delay of the longest path (Rank 0), select from the Timer menu:

```
Timer | Expand
```

In the "Expand Path Selection" submenu, enter Rank 0. The Timer will show you the delay elements as follows:

```
;  1st longest path to U1/I0:S (rising) (Rank: 0)
; Total Delay Typ Load Macro      Start pin        Net name
;  36.3    6.4 Tsu   0  DFM       U1/I0:S
;  29.9    8.0 Tpd   1  AK1       U1/I1:A          U1/N11
;  21.9    8.3 Tpd   2  NAND2     U1/I12:A         U1/N10
;  13.6  15.4 Tcq   7  DFM       U1/I8:CLK        IQ1
;  -1.8  -1.8 Psk   8             U1/I0:CLK        ICLK
```

Step 5: Create a Fuse Map to Program an FPGA

To create a fuse map which is required for programming an FPGA, select from the ALS main menu:

```
Fuse | Run
```

A fuse file CNT4LT.FUS is generated which you use for an Activator 1 or Activator 2 programmer to program the FPGA device you have selected.

To invoke a complete ALS run, you can exit the ALS menu and enter from DOS the following command:

```
alsrun CNT4LT
```

You have now completed your design.

How to Design a Complex Circuit Which Have More Pins Than the Supported PLD's

You certainly may want to use FPGA for more complex designs and not only for the trivial shown example. The design flow is however very similar to what was shown.

The only difference is that you need to break down your design into several function blocks, each of them can be fitted into a PLD, e.g. a 22V10.

Assuming you want to create a 13-bit loadable counter with 13 load inputs. This design would not fit into a 22V10. You can, however, build a 13-bit counter by cascading an 8-bit counter with a 5-bit counter.

You can use proLogic to design your own 8-bit cascadable counter or you can use the TA269 8-bit counter from the FPGA schematic library and design your own 5-bit cascadable counter.

When you use your CAD tool to connect the 8-bit counter with the 5-bit counter, the CAD tool will create a netlist for the 13-bit counter. With MAKEADL, you can then create the ADL netlist for the 13-bit counter.

Following is the source file for a 5-bit loadable and cascadable counter using proLogic:

```
title {         Filename:       cnt51.pld
                Function:       5-bit loadable counter with synchronous
                                clear, active high carry-in and carry-out
                Designer:       Dung Tu
                Date:           31.Aug.92
                Note:           22V10 is specified as device to produce a
                                PDS file and for simulation. The design is
                                later converted to FPGA using ALES.
}

include p22v10;
include XOR.H;

define  CLK = pin1;
define  CLR = pin2;
define   LD = pin3;   /* load control pin, load when LD=H */
define   CI = pin4;   /* carry-in for cascading */

/* load inputs */

define  P0 = pin5;
define  P1 = pin6;
define  P2 = pin7;
define  P3 = pin8;
define  P4 = pin9;

/* counter outputs */

define  Q0 = pin14;
define  Q1 = pin15;
define  Q2 = pin16;
define  Q3 = pin17;
define  Q4 = pin18;

define  CO = pin22;   /* carry-out for cascading */

/* 22V10 outputs defined as active-high */

Q0 = q;
Q1 = q;
Q2 = q;
Q3 = q;
Q4 = q;

/* 5-bit counter equations */

Q0.d = ( LD & P0
      | !LD & (Q0.q % CI)) & CLR;

Q1.d = ( LD & P1
      | !LD & (Q1.q % (Q0.q & CI))) & CLR;

Q2.d = ( LD & P2
      | !LD & (Q2.q % (Q1.q & Q0.q & CI))) & CLR;

Q3.d = ( LD & P3
      | !LD & (Q3.q % (Q2.q & Q1.q & Q0.q & CI))) & CLR;
```

```
Q4.d = ( LD & P4
      | !LD & (Q4.q $ (Q3.q & Q2.q & Q1.q & Q0.q & CI))) & CLR;

CO   =  !LD & Q4.q & Q3.q & Q2.q & Q1.q & Q0.q & CI;

test_vectors {

CLK CLR LD  CI  P4  P3  P2  P1  P0  Q4  Q3  Q2  Q1  Q0  CO;

C   0   X   X   X   X   X   X   X   L   L   L   L   L   L; /* clear
                                                              counter */
C   1   1   X   1   0   1   0   1   H   L   H   L   H   L; /* load 10101*/
C   0   1   X   1   0   1   0   1   L   L   L   L   L   L; /* clear     */
C   1   0   1   X   X   X   X   X   L   L   L   L   H   L; /* count 1   */
C   1   0   1   X   X   X   X   X   L   L   L   H   L   L; /* count 2   */

repeat 11 {

C   1   0   1   X   X   X   X   X   X   X   X   X   X   L; /* repeat 11
                                                                 times */
                                                       /* outputs not tested */
}

C   1   0   1   X   X   X   X   X   L   H   H   H   L   L; /* now count 14
                                                                      */
C   1   0   1   X   X   X   X   X   L   H   H   H   H   L; /* count 15  */
C   1   0   1   X   X   X   X   X   H   L   L   L   L   L; /* count 16  */
C   1   0   1   X   X   X   X   X   H   L   L   L   H   L; /* count 17  */

repeat 13 {

C   1   0   1   X   X   X   X   X   X   X   X   X   X   L; /* repeat 13
                                                                 times */
                                                       /* outputs not tested */
}

C   1   0   1   X   X   X   X   X   H   H   H   H   H   H; /* now count 31
                                                                      */
                                    /* carry-out is high because Q0 .. Q4 are high */

C   1   0   1   X   X   X   X   X   L   L   L   L   L   L; /* count 0   */

}
```

If the proLogic simulator shows no errors for each function block and the timing is not critical, your design will normally work. You can use the Timer to do a timing analysis. If your design is very complex and the timing is critical you may need to do a backannotation simulation. In this case you will need a simulator which allows you to do a postlayout timing simulation for the complete design. Please contact Texas Instruments or a Texas Instruments distributor for assisstance.

TEXAS
INSTRUMENTS